USB Communication
What is USB?

- Universal Serial Bus
- Low-Speed, Full-Speed, High-Speed: 1.5, 12, and 480 MBps
- Host-initiated Transfers
- Four transfer types: Control, Isochronous, Interrupt, Bulk
- Two transfer directions: IN & OUT (host-centric)
Control Transfers

- Typically used by USB system software to query, configure, and issue certain generic commands to USB devices.

- In PSoC, don’t need to worry about these.
Isochronous Transfers

- Guaranteed latency & bandwidth transfer
- Sorta like UDP (no error checking/correction)
- For use when data delivery time is more important than accuracy
- 0-1023 bytes per transfer with Full-speed USB
Interrupt Transfers

- Used to poll devices at regular intervals
- Typically for devices which transmit a small amount of data (mice, keyboards, etc.)
- Can also be used to signal that other endpoints have data available.
- 0-64 bytes per transfer with full-speed USB
Bulk Transfers

- Most common transfer you will use.
- 0-64 bytes per transfer with full-speed USB
- error checking
- given low priority on the bus, but generally works pretty quick
Transfer Limits

- USB divides transfers into “frames”.
- Each frame is 1 ms.
- One transaction per endpoint, per frame.
- Max bandwidth for full speed bulk endpoint = 64 bytes * 1000 = 64Kb/s.
- High-speed USB is considerably faster, as it divides each frame into 8 “microframes”, and allows for much 512 byte transfers in each microframe, allowing for 512*8*1000 = 4 MB/s.
PSoC USB Component

- Full speed USB
- Support for all 4 transfer types
- 512 Bytes of endpoint memory
- 8 Endpoints (IN or OUT or mixture of both)
Major API Components

- **USBFS_Start**
  - Starts the USB component

- **USBFS_GetEPState(uint8 epNumber)**
  - Returns state of given USB endpoint

- **USBFS_LoadInEP(uint8 epNumber, uint8 *pData, uint16 length)**
  - Loads an IN Endpoint with a data array

- **USBFS_ReadOutEP(uint8 epNumber, uint8 *pData, uint16 length)**
  - Reads data from an OUT Endpoint to a data array.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBFS_NO_EVENT_PENDING</td>
<td>The endpoint is awaiting SIE action</td>
</tr>
<tr>
<td>USBFS_EVENT_PENDING</td>
<td>The endpoint is awaiting CPU action</td>
</tr>
<tr>
<td>USBFS_NO_EVENT_ALLOWED</td>
<td>The endpoint is locked from access</td>
</tr>
<tr>
<td>USBFS_IN_BUFFER_FULL</td>
<td>The IN endpoint is loaded and the mode is set to ACK IN</td>
</tr>
<tr>
<td>USBFS_IN_BUFFER_EMPTY</td>
<td>An IN transaction occurred and more data can be loaded</td>
</tr>
<tr>
<td>USBFS_OUT_BUFFER_EMPTY</td>
<td>The OUT endpoint is set to ACK OUT and is waiting for data</td>
</tr>
<tr>
<td>USBFS_OUT_BUFFER_FULL</td>
<td>An OUT transaction has occurred and data can be read</td>
</tr>
</tbody>
</table>
IN Data flow

- PSoC checks if endpoint status is USB_IN_BUFFER_EMPTY
- PSoC loads an IN buffer with data, API sets status flag to USB_IN_BUFFER_FULL automatically
- HOST Issues an IN transaction
- PSoC USB SIE (Serial Interface Engine) responds to transaction without CPU intervention
- Endpoint status flag is set to USB_IN_BUFFER_EMPTY
- repeat....
OUT Data flow

• HOST Issues an OUT transaction

• PSoC USB SIE (Serial Interface Engine) responds to transaction without CPU intervention, sets endpoint status flag to USB_OUT_BUFFER_FULL

• PSoC checks if endpoint status is USB_OUT_BUFFER_FULL

• PSoC reads data from OUT buffer, API sets status flag to USB_OUT_BUFFER_EMPTY automatically

• repeat....
USB on the PC side

• Libraries:
  • libusb-1.0 -- best option I have found so far for cross-platform USB.
  • CyUSB -- windows-only USB driver provided by Cypress

• Finding Devices:
  • VID/PID (Vendor ID, Product ID), uniquely identifies device.
  • format is 4 hex characters per ID: 0xFFFF/0xFFFF
  • Vendor IDs are licensed by USB Consortium (but you don’t need a license to use a random one)
USB on the PC side

- **Device Descriptors**
  - Table of configuration data that tells the PC what capabilities the USB device has
    - Includes number of endpoints, type of endpoints
    - Can have multiple configurations of descriptors

- **USB Device Classes**
  - Standardized USB Class Descriptors that allow the Operating system to interface with a device without needing a custom driver
    - Examples: Mice, Keyboards, some sound cards
libusb-1.0

• C library
• Cross-platform: Mac, Linux, Windows.
• python integration with PyUSB
• other language wrappers exist as well, (.NET, etc.)
libusb-1.0 synchronous API

- `libusb_bulk_transfer(
  struct libusb_device_handle * dev_handle,
  unsigned char endpoint,
  unsigned char * data,
  int length,
  int * transferred,
  unsigned int timeout
)"
USB Endpoint conventions

• Usually written in hex.
• PSoC can handle 8 endpoints
• MSB indicates direction of endpoint:
  • 0x81-0x88 = IN Endpoints
  • 0x01-0x08 = OUT Endpoints
Assignment

• Build an application on the PSoC to send some data via USB to your PC or Mac.

• Data can be potentiometer data, button state, whatever.

• Make it do something useful! (volume control, hide browser windows, etc.)
Resources

libusb-1.0 API: http://libusb.sourceforge.net/api-1.0/index.html

LibUSBDotNET: http://libusbdotnet.sourceforge.net/V2/Index.html

For windows: ZADIG Driver installer:
https://sourceforge.net/projects/libwdi/files/zadig/

USB Overview (needs VPN)
http://proquest.safaribooksonline.com/book/-/9781435457867/usb-an-overview/ch02